DETAILED DESCRIPTION OF THE INVENTION

The present invention may include a method and system for managing a directory using structure templates. These templates specify the relationships among the various types of information in directories. Thus, a user (i.e., a person or software application interacting with a directory) can learn about the structure of the directory by consulting the directory's structure templates. Also, when attempting to modify the directory, the user is instructed by the templates as to what modifications are permitted.

One mechanism for the creation of such structure templates is through the use of a markup language such as the Extensible Markup Language (XML). XML is useful in this instance because it easily allows information about a structure to be represented in a form comprehensible both by people and computers. In accordance with the present invention, XML can be used to create the structure template describing the layout of a directory, which will in turn impose the structural requirements of that directory.

FIG. **1** shows a sub-tree **100,** of a typical directory, which is starting with an *Organization* node, of a directory created to hold information about various organizations and their parts. This directory starts with an Organizational nod **102**. Each *Organization* node **102** must have, as its children, container nodes for people (i.e., a *People* node **106**) and groups (i.e., a *Groups* node **104**) within the organization. Each *Groups* container node **104**, in turn, must contain at least a *DomainAdminGroup* node **108** and a *DomainHelpDeskAdminGroup* node **110**. Therefore, it is desirable to ensure that each time an *Organization* node is created in the directory, the requisite child nodes, along with the child nodes' requisite child nodes, if any, are also created, resulting in the sub-tree structure of FIG. **1**. In accordance with the present invention, a structure template may be created to do this.

In the example of FIG. 1, which demonstrates one of the embodiments of the present invention, the nodes of the directory tree are implemented as objects expressed in classes of the Java programming language ("Java"), which is described in *The Java Programming Language*, Ken Arnold, James Gosling, Addison-Wesley, 1996, the contents of which are hereby incorporated by reference. Java is a registered trademark of Sun Microsystems, Inc. in the United States and other countries. Other programming languages may, however, be used in other embodiments of the present invention.

Table **1** contains exemplary XML code defining the structure templates that impose the required directory structure for the example shown in FIG. 1. It contains five structure template entries. Each entry corresponds to a node that will be created in the directory tree. Lines 6-26 define the entry corresponding to an *Organization* node. Lines 31-52 define the entry corresponding to a *Groups* container node. Lines 57-73 and 78-95 define entries corresponding to *DomainAdminGroup* and *DomainHelpDeskAdminGroup* nodes respectively, creation of both of which is required when the *Groups* container is created. Lines 100-116 define the entry for the *People* container node, creation of which node is also required when an Organization node is created.

The structure template entries in Table **1** provide structural guidelines in a consistent manner. Each entry contains values for several attributes, which describe the node to be created in the directory tree. Each attribute is wrapped inside an *AttributeValuePair* tag together with its corresponding value or values. Each *AttributeValuePair* tag begins with a "<AttributeValuePair>" delimiter and ends with a "</AttributeValuePair>" delimiter, indicating the beginning and the end of the tag, respectively. The attribute name is specified inside an empty *Attribute* tag (e.g., '<Attribute name="class"/>', line 7), and each corresponding value appears inside a *Value* tag nested within the *AttributeValuePair* tag. It is possible for an attribute to contain multiple values, as is the case with the *AttributeValuePair* tag on lines 13-16 of Table **1**.

In this example, the first attribute appearing in each entry is the "class" attribute. This refers to the Java class type of the directory node that the entry corresponds to. In the first entry (Table **1**, line 8), for instance, the value "com.iplanet.ums.Organization" specifies the Java type of the object representing an *Organization* node in the directory tree.

The second attribute in each entry is the "name" attribute. This attribute specifies the name of the node in the DIT, which will be combined with the names of its ancestors to form a distinguished name. In Table **1**, this attribute appears in the form of an equation. When a user adds a node to the directory tree, the right-hand side of this equation will be replaced by the value of the attribute 'o', which will be available to the user creating the new node. If, for example, the user creates a node for an organization named "acme", the name of the node will be "o=acme".

Next is an attribute which appears only in entries for nodes whose creation requires the automatic creation of at least one child node. This attribute, called the "childNode" attribute, identifies the entries for the children whose creation is required. The childNode attribute provides ordered links and parent-child relations between individual entries to form a composite object. Lines 14-15 of Table **1** indicate that nodes for "OrganizationalUnit" and "PeopleContainer" should be created under the Organization entry. Similarly, lines 39-40 indicate that the creation of the Organizational Unit node requires the creation of two child nodes.

The "template" attribute identifies a "creation template" that is used to create an entry. The creation template specifies parameters that provide ground rules for creating an entry of that type. The name of the creation template is needed by a process that creates the entry. For example, the creation template used for creating an *Organization* entry, is called "BasicOrganization".

The "filter" attribute, which appears next, is used to resolve ambiguity if there is more than one template defined with the same class name. It is possible for two or more sibling nodes (i.e., nodes with the same parent node) to have the same class type and name, since it may be desirable to use different structure template entries depending on criteria such as, say, the current date. For instance, it may be desirable to use a different structure template in creating a *Groups* container node for each month of the year. In that case, the system administrator can set up twelve structure templates with the same template name and "class" attribute but different "filter" attribute values. The administrator can define the template entries so that the value of the "filter" attribute is the name of the month during which that template is to be used. A user can then determine, based on the value of the "filter" attribute, which template to use.

The "priority" attribute is used to resolve ambiguity if there are multiple templates defined with the same name and filter. This is a useful feature because it allows the system administrator to keep multiple versions of the same template in the system, and then choose which version should be in active use simply by adjusting the "priority" attribute values of the different versions. In one embodiment, the version with the lowest priority is the one in current use.